

# Algoritmos y Estructuras de Datos II

## Recuperatorio 1er Parcial – Miércoles 10 de Julio de 2024

LU	Apellido y Nombre	#Hojas	E1	E2	E3	E4	Nota	Corrigió
		5	20	30	20	30	100	win

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, más los dos apuntes de TADs
- Cada ejercicio debe entregarse en **hojas separadas**
- Incluir en cada hoja el número de orden asignado, número de libreta, número de hoja, apellido y nombre
- El parcial se aprueba con 70 puntos
- Justifique todas las respuestas

### E1. Especificación de problemas [20 pts]

Se desea especificar el problema *secuenciaConMasPares*, que dada una secuencia de secuencias de enteros, devuelve el elemento que contenga más números pares. En caso de empate se puede devolver cualquiera.

#### Ejemplos

- *secuenciaConMasPares*( $\langle \langle 1, 2, 3 \rangle, \langle 4, 5, 6 \rangle, \langle 7 \rangle, \langle 8, 10 \rangle \rangle$ ) =  $\langle 4, 5, 6 \rangle$
- *secuenciaConMasPares*( $\langle \langle 1, 2, 3 \rangle, \langle 4, 5, 6 \rangle, \langle 7 \rangle, \langle 8, 10 \rangle \rangle$ ) =  $\langle 8, 10 \rangle$

Para esta secuencia de entrada cualquiera de las dos salidas es correcta.

### E2. TADs [30 pts]

La masividad de la materia Algoritmos y Estructura de Datos (AED) en el primer cuatrimestre de 2024 generó que el primer parcial de la materia se tuviera que tomar en el aula más grande disponible en la Facultad (Magna del Pab2). Esta aula cuenta con 2 puertas, una a cada lado del aula. Debido a la poca previsión de los docentes (que no se previeron organizar un sistema de ingreso ordenado al aula), los y las estudiantes aguardaron el ingreso al aula en ambas puertas. Llegada la hora del inicio del parcial, para poder ordenar el ingreso, uno de los Profesores decidió dejar pasar a los estudiantes de a uno a la vez, alternando un ingreso de cada puerta.

Se desea especificar el TAD *dobleCola* $\langle T \rangle$  que modele un sistema como el descrito anteriormente: existen dos colas, las mismas se encolan de forma independiente, y la salida de las mismas (desencolar) se dá de forma alternada. Para dicha especificación sólo se pueden usar tipos básicos de especificación: Z, R, bool, seq, string, tupla, conj, dict.

- Elegir los observadores y especificar los procs necesarios.
- ¿Es necesario escribir el predicado de igualdad? Si es necesario escríbalo. Si no, explique por qué.
- Explicar con palabras qué habría que modificar en el TAD para que en lugar de tener 2 colas se tienen  $n$  colas, donde el valor de  $n$  se indica al crear la instancia del TAD.

### E3. Precondición más débil [20 pts]

Dado el siguiente condicional determinar la precondición más débil que permite hacer valer la poscondición (Q) propuesta.

```

F1(inout A: seq<int>, in i: int)
  p := i+1
  A[p] := 2*p

```

$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \bmod 2 = 0)$

- Describir en palabras la WP esperada
- Derivarla formalmente a partir de los axiomas de precondición más débil. Para obtener el puntaje máximo deberá simplificarla lo más posible.

**E4. Correctitud del ciclo [30 pts]**

Dado el siguiente programa con su especificación

$$P \equiv \{|s| \bmod 2 = 0\}$$

$i := 0$

$\text{suma} := 0$

```
while (**COMPLETAR**) {  
    **COMPLETAR**  
}
```

$\text{res} = \text{suma} > 5$

$$Q \equiv \{\text{res} = \text{True} \leftrightarrow \sum_{j=0}^{|s|-1} s[j] > 5\}$$

Contamos con el siguiente invariante, que sabemos que es correcto:

$$I \equiv \{\text{EnRango}(i) \wedge \text{suma} = \sum_{i=0}^{i-1} s[i] + \sum_{i=|s|-i}^{|s|-1} s[i]\}$$

- Escriba el predicado *EnRango*) y complete el código en base al invariante y la especificación.
- Explicar con sus palabras, sin demostrar formalmente, por qué éste invariante cumple con los 3 primeros puntos del teorema del invariante.
- Proponer una  $f_v$  y demostrarla formalmente.



① Problema secuencia con mas pzes, que dada una secuencia de secuencias de enteros, devuelve el elemento que contenga mas pzes.

pred esPzr( $n: \mathbb{Z}$ ) {  
 $n \bmod 2 == 0$  }

Aux CantidadPares ( $s: \text{seq}(\mathbb{Z})$ ) :  $\mathbb{Z}$  {  
 $\sum_{j=0}^{|s|-1} ( \text{if } \text{esPzr}(s[j]) \text{ then } 1 \text{ else } 0 \text{ fi} )$

~~Finalmente~~ debo contemplar el caso de la lista vacía, y de una lista que tenga como elementos 2 listas vacías y 10 ~~secu~~ impares. Si todas las listas ~~secu~~ dentro de  $s$  no tienen pzes, devuelvo cualquiera de ellas.

Si mi lista inicial es  $|s|=0$ , como no tengo elementos, no puedo ejecutar este problema. Si  $|s|=1$  y su único elemento es la lista vacía, si puedo porque existe un elemento en  $s$ . ~~Si no es así, no puedo.~~

Tengo que describir lógicamente que el resultado cumple

- Ser un elemento de  $s$  (por eso no excluir  $|s|=0$ )
- Para todo otro elemento de  $s$ , la cantidad de pzes debe ser menor o igual a la cantidad de pzes del resultado.

proc secuencia con mas pzes (in  $s: \text{seq}(\text{seq}(\mathbb{Z}))$ ) :  $\text{seq}(\mathbb{Z})$  {

requiere {  $|s| > 0$  }

asegura {  $\text{res} \in s$  }

asegura {  $(\forall t: \text{seq}(\mathbb{Z})) (t \in s \wedge t \neq \text{res} \rightarrow$

$\text{CantidadPares}(\text{res}) \geq \text{CantidadPares}(t)$  }

}



- ② TAD  $\text{dobleCola}(T)$  • debe tener 2 colas
- Al desencolar debe intercambiar ambas colas.
  - se encolan independientemente entre.

• Las colas se desencolizan con lógica FIFO, por lo que necesito representárselas con una tipo básica que me permita eliminar sin saber cuál es, pero sí su posición.

Para las colas puedo usar secuencias donde cuando encolamos a un elemento lo encolamos al final usando  $\text{concat}$ , y al desencolar puedo decir que el primer elemento de la cola vieja no está en la nueva. (usando  $\text{slice}$ ).

```
TAD  $\text{dobleCola}(T)$  {
  obs colas : seq  $\text{seq}(T)$ 
  obs ultimaDesencolada : int  $\mathbb{Z}$ 
}
```

Guardo las colas en una secuencia para poder acceder fácilmente a la que debo desencolar. En  $\text{ultimaDesencolada}$  guardo el índice de la última cola que desencolé. Así, cuando llamo al proc  $\text{desencolar}$ ,  $\text{ultimaDesencolada} = \text{len}(\text{dobleCola.colas}) - 1$ , desencolo en  $\text{dobleCola.colas}[\text{ultimaDesencolada}]$ , y si no, desencolo en  $\text{old}(\text{dobleCola.colas})[\text{ultimaDesencolada} + 1]$ .

Para este TAD voy a tener que definir los siguientes procs.

```
proc  $\text{nuevoDobleCola}()$  :  $\text{dobleCola}(T)$ 
  proc  $\text{encolar}(inout c : \text{dobleCola}(T), in i : \mathbb{Z}, in e : T)$ 
    donde paso el  $i$  que es un  $\mathbb{Z}$  a la cola donde quiero encolar
  proc  $\text{desencolar}(inout c : \text{dobleCola}(T)) : T$ 
```

```
TAD  $\text{dobleCola}(T)$  {
  obs colas :  $\text{seq}(T)$ 
  obs ultimaDesencolada :  $\mathbb{Z}$ 
```

```
proc  $\text{nuevoDobleCola}()$  :  $\text{dobleCola}(T)$  {
  requiere {true}
  asegura {res.colas =  $\langle \rangle, \langle \rangle$ }
  asegura {res.ultimaDesencolada = 1}
}
```

```
proc  $\text{encolar}(inout c : \text{dobleCola}(T), in i : \mathbb{Z}, in e : T)$  {
  requiere {  $0 \leq i < \text{len}(c.colas)$  }
  requiere {  $(\forall col : \text{seq}(T)) (col \in c.colas \rightarrow e \notin col)$  }
  asegura {  $(\forall col : \text{seq}(T)) (col \in c.colas \rightarrow e \notin col)$  }
  asegura {  $(\forall j : \mathbb{Z}) (0 \leq j < \text{len}(c.colas) \wedge j \neq i \rightarrow c.colas[j] = \text{old}(c).colas[j])$  }
  asegura {  $(\forall j : \mathbb{Z}) (0 \leq j < \text{len}(c.colas) \wedge j \neq i \rightarrow c.colas[j] = \text{old}(c).colas[j])$  }
```

```
asegura {  $c.colas[i] = \text{concat}(\text{old}(c).colas[i], \langle e \rangle)$  }
```

```
asegura {  $\text{len}(c.colas) = \text{len}(\text{old}(c).colas) + 1$  } (5. que se res)
```



~~Procedimiento para desenrollar~~

asegura { c.ultimaDesenrollada = old(c).ultimaDesenrollada }

Proc desenrollar (invar c: dobleCola < T): T {

requiere { (c.ultimaDesenrollada = 1  $\wedge$  c.colas[0]  $\neq$  <T>)  $\vee$   
c.ultimaDesenrollada = 0  $\wedge$  c.colas[1]  $\neq$  <T> ) }

~~asegura { c.ultimaDesenrollada = 1 }~~

asegura { |old(c).colas| = |c.colas| }

asegura { c.colas[old(c).ultimaDesenrollada] = ~~old(c).colas[old(c).ultimaDesenrollada]~~  
old(c).colas[old(c).ultimaDesenrollada] }

asegura { c.ultimaDesenrollada = proxima(old(c)) }

asegura { c.colas[c.ultimaDesenrollada] =

subsea ( old(c).colas[c.ultimaDesenrollada], 1, old(c).colas[c.ultimaDesenrollada] ) }  
asegura { res = old(c).colas[c.ultimaDesenrollada] [ |c.colas[c.ultimaDesenrollada]| ] }

}

Aux proxima (c: dobleCola < T): T {

~~if~~ (if c.ultimaDesenrollada == 1 then 0 else 1 }

Comentarios:

Proc nuevaDobleCola: fongo que ultimaDesenrollada sea 1 por  
que cuando desenrola por primera vez lo haga en la lista 0.

Proc enrollar: Asumo que no tiene sentido enrollar a un elemento en  
dos colas distintas, ni 2 veces en la misma cola.  
• Cuando uso concat, me garantiza que todo el en la  
la secuencia vieja está en la nueva y que (como uso concat y agregación  
el elemento al final) también mantienen su posición. También  
puedo asumir que el último elemento del resultado es el elemento  
que me pasan por parámetro.

Proc desenrollar: Asumo que no tiene sentido desenrollar en una lista  
vieja

• Subsea garantiza que todos los elementos de la  
secuencia vieja, ~~están~~ que están entre las posiciones inclusive  
y |secuencia| exclusive, van a estar en el resultado de  
subsea(secuencia, i, |secuencia|) y ~~están~~ van a estar ordenados  
como en la original.

Como en el problema del punto 2a ya sé que tengo 2  
puertos, puedo usar el ~~aux~~ proxima y que el require de  
desenrollar sea válido.

secuencia de

b) Como ya defini mi TAD con un observador secuencias ~~secuencia~~  
~~secuencia~~, y me importa el orden de sus elementos tanto  
por la secuencia c.colas como para cada subsecuencia de c.colas,  
y mi otro observador es un entero, no necesito definir un



p587. Hoja 3 (EJ 2)

predicador de igualdad observacional. 2 secuencias son iguales entre sí solamente si son iguales elemento por elemento, por lo que deben tener la misma longitud; y  $(\forall j: 2) (0 \leq j < |S| \rightarrow S[j] = T[j])$ .

Por el uso que le doy a la secuencia, siempre me zanca con igual los observadores colts.

Por ejemplo, veo que  $\langle \langle 1, 2 \rangle \langle 3 \rangle \langle 4 \rangle \rangle \models \langle \langle 3 \rangle, \langle 1, 2 \rangle \langle 4 \rangle \rangle$   
porque desencolar (si no desencolée nunca antes) da resultados distintos.

Por el observador `ultimaDesencolada` también me zanca con igual porque si en 2 instancias de `TAD` estos difieren, desencolar me da a dar el mismo resultado.

Ⓢ si en lugar de tener 2 colts se tienen  $n$  colts, debería cambiar la siguiente.

En el proc `nuevaDobleColt`, voy a pasar como parámetro  $n$  y,   
res. colts =  $\lceil \frac{1}{2} * n \rceil$  (tengo  $n$  secuencias vecinas). Además,   
debo poner res. `ultimaDesencolada` =  $n - 1$ .

En el proc `Encolar` no debo cambiar nada (es genérico a longitud de c.colts)

En `desencolar`, voy a tener que decir que por cada colt distinto a 12 que voy a desencolar, ~~que se mantiene igual antes y después del proc.~~ <sup>no</sup> El ~~proximo~~ <sup>proximo</sup> va a tener que entender los casos donde `c.ultimaDesencolada`  $< n - 1$  (en esos casos se suma 1) y el caso donde `c.ultimaDesencolada` =  $n - 1$  (donde se reestablece en 0). El requisito va a necesitar que `c.colts [proximo]`  $\neq \langle 7 \rangle$ .  
~~Por último, voy a necesitar asegurarme que~~

Weg de escribir el requisito usando el ~~proximo~~ <sup>ante</sup> `proximo`, desencolar se vuelve genérico por cada longitud de c.colts porque cambio el `c.ultimaDesencolada` ~~antes~~ a la colt en la que me toca desencolar antes de desencolar en ella.



- ③ Determinar la precondición más débil que permite hacer la postcondición  $Q$ . Véase.

$f_1(\text{input } A: \text{seq}(\text{int}), \text{in } i: \text{int})$

$p = i + 1$

$A[p] = 2p$

$Q \equiv (\forall j: \mathbb{Z}) (0 \leq j < |A| \rightarrow A[j] \bmod 2 = 0)$

- ④ ~~para este in~~

Para el valor  $i$  que yo pongo como parámetro, se que la posición  $SL[i+1] = 2 \cdot (i+1)$ , lo que automáticamente la hace par.

En la WP espero que el rango válido de  $i$  sea  $-1 \leq i \leq |A|-2$  y que, para todo  $j$  que sea distinto de  $i+1$ , el elemento ya sea par.

- ⑤  $WP(f_1, Q) \equiv WP(p := i+1; A[p] = 2p, Q) \equiv WP(p := i+1, WP(A[p] = 2p, Q))$

$WP(A[p] = 2p, Q) \equiv \text{wp}(A[p] = 2p, Q) \equiv$

$0 \leq p < |A| \wedge Q \stackrel{A[p]=2p}{\text{at. 2}}$

$0 \leq p < |A| \wedge (\forall j: \mathbb{Z}) (0 \leq j < |A| \wedge j \neq p \rightarrow A[j] \bmod 2 = 0)$

(como  $A[p] = 2p$ , siempre es par, entonces no tengo que pedir que lo sea en la WP.

$WP(p := i+1, 0 \leq p < |A| \wedge (\forall j: \mathbb{Z}) (0 \leq j < |A| \wedge j \neq p \rightarrow A[j] \bmod 2 = 0)) \equiv$

$0 \leq i+1 < |A| \wedge (\forall j: \mathbb{Z}) (0 \leq j < |A| \wedge j \neq i+1 \rightarrow A[j] \bmod 2 = 0) \equiv$

$(-1 \leq i < |A|-1 \wedge (\forall j: \mathbb{Z}) (0 \leq j < |A| \wedge j \neq i+1 \rightarrow A[j] \bmod 2 = 0))$

esto es el resultado de la WP.



1. (4) (2)

$$I \equiv \{ \text{enRango}(i) \wedge \text{suma} = \sum_{j=0}^{|S|-1} S[j] + \sum_{i=|S|-i}^{|S|-1} S[i] \}$$

$$\text{pred enRango}(i) \{ 0 \leq i \leq |S|/2 \}$$

1.  $i$  debe estar en el rango indicado porque cuando  $i = n/2$ , ya voy a haber sumado los valores correspondientes en otras iteraciones.

El programa completo quedaría

$i := 0$

$\text{suma} := 0$

while  $i < |S|/2$  do

$\text{suma} := \text{suma} + S[i] + S[|S|-1-i]$

$i := i + 1$

end while

$\text{res} = \text{suma} > 5$

b)  $p \rightarrow 1$  se cumple porque cuando  $i=0$ , las sumatorias se van de rango.

$\{i \wedge B\} \wedge \{i\}$  se cumple porque, si respecto la guarda del while, como primero le sumo una  $2i$  (cuando hago  $|S|/2$ ), coinciden las sumatorias con la  $\text{suma} + S[i] + S[|S|-i-1]$  (en  $i+1$ )

$i \wedge \neg B \rightarrow Q_c$

$$Q_c \equiv \{ \text{suma} = \sum_{j=0}^{|S|-1} S[j] \wedge i = |S|/2 \}$$

Se cumple porque  $i \wedge \neg B$  cumple cuando  $i = |S|/2$ .

c) Probarlo como  $f_v = |S|/2 - i$

$$\{i \wedge B \wedge v_a = |S|/2 - i\} \rightarrow \text{wp}(S, \text{for } i < v_a) \equiv$$

$$i \wedge B \wedge v_a = |S|/2 - i \rightarrow \text{wp}(\text{suma} \dots; (|S|/2 - i < v_a)_{i+1}^i)$$

$$i \wedge B \wedge v_a = |S|/2 - i \rightarrow \text{wp}(\text{suma} \dots; |S|/2 - i - 1 < v_a)$$

Como sabemos que  $\text{suma}$  no modifica en  $|S|/2 - i - 1$  ni  $v_a$ , sería como poner un skip en su lugar. ~~como en el caso anterior~~



①  $\text{enlargo}(i) \wedge i < |s|/2$  ~~es verdadero~~  $\wedge \text{suma} = \sum_{i=0}^{i-1} s[i] + \sum_{i=|s|-i}^{|s|-1} s[i]$   $\wedge$

$\Rightarrow v_0 = |s|/2 - i \rightarrow v_0 > |s|/2 - i - 1$   $\quad ?$

¿suma que la que hay delante del índice es verdadero?

$v_0 = |s|/2 - i$

$|s|/2 - i > |s|/2 - i - 1$   
 $a) > -1$  siempre.

Podemos decir que la implicación es válida.

②  $i \wedge \text{fr} \leq 0 \rightarrow \neg b?$

$\text{enlargo}(i) \wedge |s|/2 - i \leq 0 \leftrightarrow i = |s|/2$

$\text{enlargo}(i) \wedge \text{fr} \leq 0 \wedge |s|/2 - i \leq 0 \wedge \text{suma} = \sum_{i=0}^{i-1} s[i] + \sum_{i=|s|-i}^{|s|-1} s[i] \rightarrow$   
 $\neg (i < |s|/2) ?$

$i = |s|/2 \rightarrow i > |s|/2$  es verdadero siempre.

~~Podemos~~ podemos decir que vale la implicación.

Acabamos de probar que nuestro ciclo termina gracias al teorema de terminación. Para eso debíamos proponer un  $\text{fr}$  que sería correcto en el caso donde nuestro ciclo termine. Por esto puedo decir que la  $\text{fr}$  propuesta es correcta.